7th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS
S u c e a v a, R o m a n i a, M a y 27 – 29, 2 0 0 4

HARETICK: A REAL-TIME COMPACT KERNEL FOR CRITICAL APPLICATIONS ON EMBEDDED PLATFORMS[1]

Mihai V. MICEA
Department of Computer Science and Engineering
POLITEHNICA University of Timisoara
2, Vasile Parvan Bv., 300223 – Timisoara, Romania
Tel: +40 256 403271, Fax: +40 256 403214
micha@dsplabs.utt.ro

Abstract. *This paper discusses the problem of designing and implementing a real-time compact kernel for embedded and DSP-based platforms, able to provide a fully predictable execution environment for critical applications. Based on a sound and uniform set of models defined for time, signals and tasks, we describe the architecture and operating principles of a particular real-time kernel – "HARETICK". Its modular and compact architecture is designed around the key idea of enabling concurrent execution of hard real-time (HRT) and soft real-time (SRT) tasks in two separate contexts. The HRT execution context is based on non-preemptive scheduling algorithms and has precedence over the SRT context, which uses traditional, preemptive, priority-based scheduling techniques.*
Key words: *real-time, embedded, operating kernel, non-preemptive, scheduling, temporal parameters*

Introduction

Embedded systems and digital signal processing (DSP) systems [1], [2], are widely used in today's digital control applications, requiring in most cases real-time behavior of the hardware-software components. Many applications have a critical impact on the environment and/or on the human factor with which they interact. Examples of such applications include: modern flight control systems, fly-by-wire, auto-pilot, navigation equipment, rocket control and guiding, military communication equipment, automotive control, industrial mechatronics, and nuclear plant surveillance and control systems.

There are two essential characteristics a hardware-software platform has to meet in order to provide correct operation results for critical applications [3], [4], [5], [6] and [7]:

(a) The entire process of system/application development should include the time coordinate,
(b) The system must provide maximum of predictability for the hard real-time tasks.

Although a very large number of projects have been lately developed in the field of real-time and embedded systems, both in the industry and the academic communities, there still are many important issues to be addressed.

A large number of real-time systems are still being developed as ad-hoc implementations and oriented mainly towards particular applications – especially if they have critical impact on the environment. Many other real-time systems derive from traditional, time-sharing architectures, further adapted to real-time applications and optimized in order to increase their speed of reacting to events. Their main disadvantage resides on the lack of compatibility between their hardware/software architecture, designed to provide a good *average case* behavior, and the requirements of real-time applications that specify a correct behavior of the system even in *worst case operating conditions*.

Another important issue regarding the predictability of hard real-time systems is related to the unrestricted use of interrupts [7] and the associated asynchronous mechanisms and tasks.

[1] This work is supported in parts by a grant from Motorola, Incorporated, through the PhD Collaboration Program "DALT-PhD/2001", with the Department of Computer Science and Engineering (DCSE) Timisoara.

Our current research focuses on developing suitable methodologies and architectures that enable hard real-time systems to meet the two basic requirements previously stated in this section. The approach is based on studying and integrating proper models of time, signals and tasks, emphasizing on non-preemptive scheduling techniques.

This paper discusses the problem of designing and implementing a real-time compact kernel for embedded and DSP-based platforms, able to provide a fully predictable execution environment for critical applications.

An Operating Environment for Real-Time Applications

Our approach on designing and implementing real-time systems for critical applications is based on the following key ideas:

(i) Based on the general acceptance that even the critical applications contain both types of tasks – soft real-time (*SRT*) and hard real-time (*HRT*) tasks, the host platform must accommodate properly the concurrent execution of the two types of tasks.

(ii) *The use of interrupts and asynchronous mechanisms* in the system generates predictability problems, affecting its capability to guarantee that the temporal specifications of hard real-time tasks can be met, in any operating conditions [3], [5], [7].

(iii) In order to provide maximum predictability for HRT tasks, *non-preemptive models and techniques* are been studied and used for scheduling and executing hard real-time and critical tasks [8], [9].

(iv) The entire process of system/application development should integrate the time coordinate within homogenous methods and models for each of its phases [5], [6].

(v) For HRT systems design and implementation, the offline analysis of HRT and critical tasks, prior to their execution on the system, is an imperative requirement [6].

(vi) Structures and mechanisms that generate unpredictability in the system operation, such as: interrupts, cache and virtual memories, pipelining, cycle stealing DMA, recursive function calls, unbounded program loops, dynamic resource allocation, must be avoided [3], [4], [5].

Figure 1 depicts the OPEN-HARTS system (*Operating Environment for Hard Real-Time Systems*), composed of two main subsystems.

INVERTA (*Integrated Visual Environment for Real-Time Application Analysis and Development*), provides the programmer with the necessary tools for designing, specifying, programming, validating and analyzing the applications on a host (mobile) computer.

HARETICK (*Hard Real-Time Compact Kernel*), which runs on the target platform and provides two distinct execution contexts, operating concurrently: a non-preemptive context for HRT tasks, along with a traditional, preemptive context for SRT tasks.

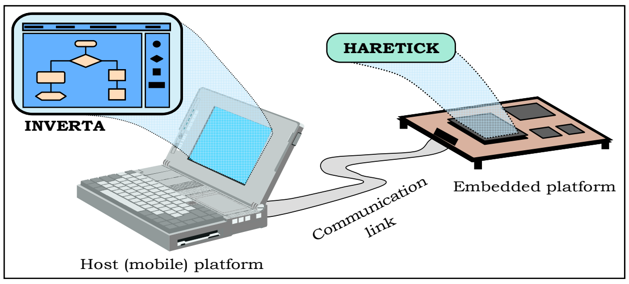

Figure 1. General architecture of the OPEN-HARTS system

After some real-time application has been successfully developed and analyzed within the INVERTA, it is loaded on the target platform to be executed with the necessary predictability under the HARETICK kernel.

The HARETICK Kernel: Characteristics and Components

HARETICK is a single-user, multitasking, hybrid real-time operating kernel for embedded and DSP-based platforms, designed to provide maximum predictability to critical or hard real-time applications. "Hybrid real-time" refers to the fact that the kernel provides support for two concurrent task execution environments: the HRT context, for the execution of hard real-time tasks in a non-preemptive manner, and the SRT context, for the execution of soft real-time (or regular) tasks in a classical, preemptive and priority-based manner. Therefore, HARETICK is able to guarantee that all the tasks scheduled and executed within the HRT context will meet all their temporal specifications, even in the worst case operating conditions.

As a consequence of the features mentioned above, HARETICK currently allows only one interrupt source: the *Real Time Clock* (*RTC*).

Figure 2 presents the main components of the kernel and their relationship.

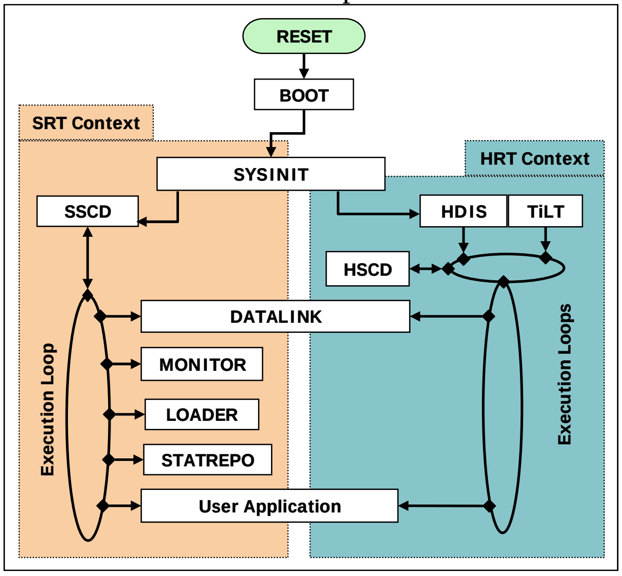

Figure 2. Main HARETICK components

At system startup (after RESET), the BOOT sequence is launched, thus loading the other kernel components into memory. Then, system initialization is performed. As seen from Figure 2, the SYSINIT task belongs to the SRT context (in fact, it is the first SRT task of the system). SYSINIT also starts the HRT execution context, by activating the RTC interrupts. Before termination, SYSINIT calls the SRT context scheduler (SSCD) which will take over the scheduling and execution of the SRT tasks in the system.

The execution within the HRT context starts with the system dispatcher/executive (HDIS), which is activated by each of the RTC interrupt events. The first hard real-time task HDIS launches is the HRT scheduler (HSCD). It uses non-preemptive algorithms optimized for embedded platforms (a modified version of the EDF – Earliest Deadline First approach [8], [9]), to fill in a Dispatch Table with HRT tasks and their calculated start times, in a cyclic manner.

DATALINK is a hybrid task (it contains both SRT and HRT components) for managing the communications link of the HARETICK to a host computer. After being successfully designed, programmed and analyzed within the INVERTA environment, a new application can be loaded on the target platform to be executed. The LOADER task is responsible for creating the structures needed to represent the application into the kernel. It can also be viewed as the application memory manager. The kernel user interface is implemented by the task MONITOR.

The kernel can provide various status and execution reports upon request. STATREPO is responsible for gathering the necessary data from the system and for generating the reports.

TiLT (Time Log Tool) is a subroutine attached to the system executive HDIS, which stores in a dedicated memory buffer information regarding each execution of HRT tasks.

Time Management and Representation

Time is a key operating dimension for real-time systems and it must be considered in all the development stages of such systems: formal specification and verification, programming, analysis, scheduling and execution [10], [11].

A model of time, suitable for embedded and DSP-based platforms, can be defined based on the characteristics of system clock generating devices (here, the *Real Time Clock*, *RTC*). Thus, the system temporal domain has a linear and discrete structure, and is limited to the left (i.e. there exists the initial time instant, $t_0 = 0$, corresponding to the system startup moment in the absolute time domain). The time unit corresponds to an interval Δt_{RTC} from the absolute time domain (Figure 3).

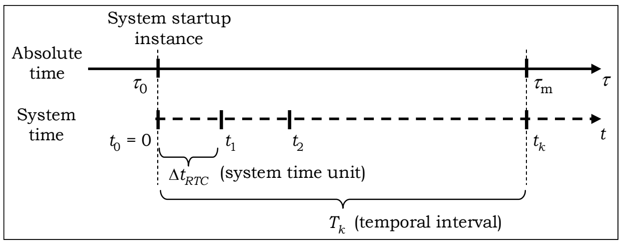

Figure 3. Absolute time and system time

The system time model has a metric, thus allowing one to express quantitative relationships between time instances or intervals, and to calculate the length (duration) of the time intervals, as in (1).

$$\begin{cases} T_k = \tau_m - \tau_0 = k \cdot \Delta t_{RTC} & \text{absolute time} \\ T_k = t_k - t_0 = k & \text{system time} \end{cases} \quad (1)$$

A correspondence between the absolute and the system time domains can be stated as in (2).

$$\tau_i = \tau_0 + (t_i - t_0) \cdot \Delta t_{RTC} \quad (2)$$

The real-time applications designed to run on the HARETICK kernel use the time model described above. The importance of time is emphasized on the kernel by the fact it implements three distinct structures for time management: the "Absolute Time" variable (`Sys_AbsTime`), the "Scheduling Time" variable (`HScd_TSched`) and the Real Time Clock device (RTC), composed of two cascaded timers.

Figure 4 presents the time management structures in HARETICK for an implementation with the Motorola DSP56307, a 24-bit digital signal processor [12], [13].

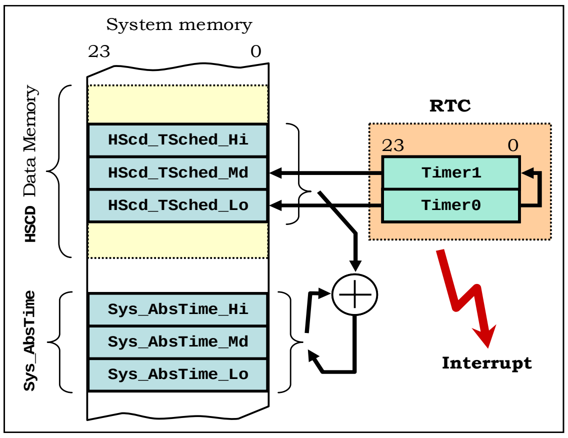

Figure 4. Time management structures and mechanisms in the HARETICK kernel

The RTC measures the system real time and, once started (by the SYSINIT task), it runs in a continuous and independent manner. The RTC timers are programmed by the system executive (HDIS) to generate compare interrupts at particular time instants, corresponding to the start moments of scheduled HRT tasks.

The "Scheduling Time" is used by the HRT scheduler (HSCD) to compute the execution schedule of the HRT context in a cycling manner. At each execution of the HSCD (corresponding to the start of a new scheduling cycle), it also updates the system "Absolute Time" with the duration of the previous cycle.

Hard Real-Time Task Representation: the ModX

Assuming the non-preemptive operation, we introduce a particular model for HRT tasks, which can be used in real-time system specification, analysis and execution.

A *ModX* (*e<u>x</u>ecutable <u>mod</u>ule*) is defined as a periodic, modular, HRT task, with complete and strict temporal specifications, scheduled and executed in non-preemptive context:

$$M_i \equiv \langle \boldsymbol{T}, \boldsymbol{P}, \boldsymbol{S}, \boldsymbol{F} \rangle \quad (3)$$

where: $\boldsymbol{P} = \{\boldsymbol{P}_{IN}, \boldsymbol{P}_{OUT}, \boldsymbol{P}_{GLB}\}$ is the set of input, output and global parameters of M_i, respectively; $\boldsymbol{S} = \{\boldsymbol{S}_{IN}, \boldsymbol{S}_{OUT}\}$ is the set of input and output signals which M_i interacts with; $\boldsymbol{F}$ is the task's instruction set (its functional specification); and:

$$\boldsymbol{T} = \left\{ T_{pr}^{M_i}, T_{ex}^{M_i}, T_{dl}^{M_i}, T_{dy}^{M_i}, N^{S_j} \right\} \quad (4)$$

represents the set of temporal parameters of M_i, in their respective order: period, execution time, deadline, delay of execution during each period, and execution count (see also Figure 5).

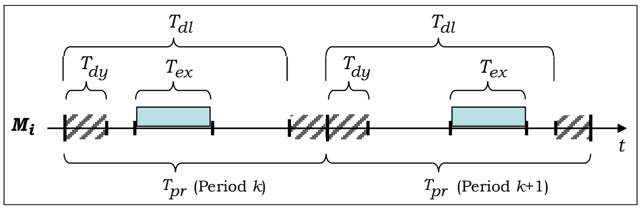

Figure 5. Temporal parameters of ModX M_i

The non-preemptive approach of modeling each HRT task of an application as a ModX, requires actual values (in system time units, see (1) and (2)) for a minimum of temporal parameters, such as the period, execution time and execution count. These values are set or calculated during the application specification and analysis phases, and will be verified during the validation phase. These phases must be performed prior to the actual scheduling and execution of the application [6], in order to ensure maximum predictability. Two basic relationships between the temporal parameters of any ModX that must be verified are given in (5) and (6):

$$0 < T_{ex}^{M_i} \le T_{dl}^{M_i} \le T_{pr}^{M_i} \quad (5)$$

$$0 \le T_{dy}^{M_i} \le T_{dl}^{M_i} - T_{ex}^{M_i} < T_{dl}^{M_i} \le T_{pr}^{M_i} \quad (6)$$

The formulas basically state that the execution time is a positive, non-zero value and it must be less than the deadline and the period of the ModX.

The execution time is considered to be a constant value during the entire task operation, and to be equal to the task's *WCET* (*Worst Case Execution Time*) that results from the program timing analysis [5]. The modular approach of defining the HRT task model – the ModX, enables automatic techniques of WCET estimation during application analysis.

From the structural and from the functional points of view, the ModX is a standalone software module, similar to the "basic block" element used in the compilers theory, and which is executed in non-preemptive context. Therefore, the ModX implements *atomic operations*, eliminating the need of synchronization of concurrent access to shared resources of the system or the application.

Information exchange between ModXs is performed through the input, output and global parameters.

As asynchronous mechanisms have been eliminated from the model, input signals are processed by their corresponding ModXs by periodic polling techniques.

While a ModX M_i, once loaded on the target platform, is scheduled for as long the application is running, its execution count parameter, N^{M_i}, specifies three possibilities for the effective execution of M_i:

- $N^{M_i} = \infty$, states continuous execution of M_i (i.e. the ModX will be executed each time it is scheduled);
- $N^{M_i} = 0$, specifies that M_i will not be executed, although currently scheduled. This type of ModX is called a *Ghost ModX*;
- $0 < N^{M_i} < \infty$, specifies that Mi will be executed at the time instance corresponding to the schedule. Before execution though, the kernel executive task decrements the execution count of M_i.

An interesting feature of the ModX model is that its execution count (and, therefore, its effective execution) can be controlled (changed) at runtime by other ModXs or even by itself.

Basic Kernel Operation

Our discussion in this paper focuses on the particularities of operation of the HRT tasks (the ModXs) within the HARETICK kernel.

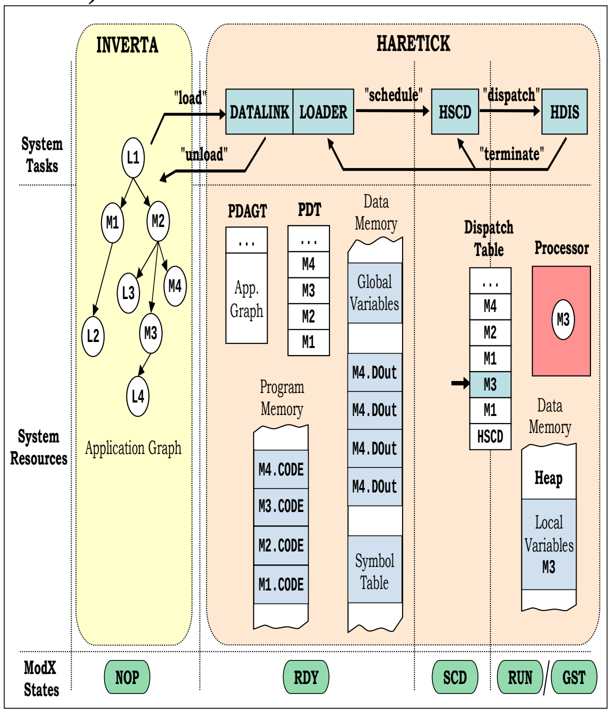

Figure 6. The ModX states and the general operation of the kernel

Figure 6 depicts the states of ModXs belonging to an application during its operation within the kernel.

The application is developed and analyzed on a host (mobile) computer within the INVERTA environment. Its ModXs are unknown to the HARETICK kernel and their status is defined as "*NOP*" (*No Operation*).

Loading of the application onto the target platform is performed by the LOADER task, through the communication interface provided by the DATALINK task. The LOADER defines the following structures for representing the application and its ModXs:

- The Program Directed Acyclic Graph Table (PDAGT) describes the control and data dependences of each ModX;
- The Process Descriptor Table identifies each ModX along with its parameters (including the temporal behavior, see (4));
- The compiled code area (`Mi.CODE`);
- The output parameter area (`Mi.DOut`);
- The application's global parameter area;
- The symbol table.

After all the ModXs have been successfully loaded into the system, they are in the "*RDY*" state, meaning they are *Ready for Scheduling*.

The HRT scheduler (HSCD) applies particular non-preemptive scheduling algorithms to fill the Dispatch Table with ModX identifiers and their corresponding starting times, thus defining a *scheduling cycle*, each time it is executed.

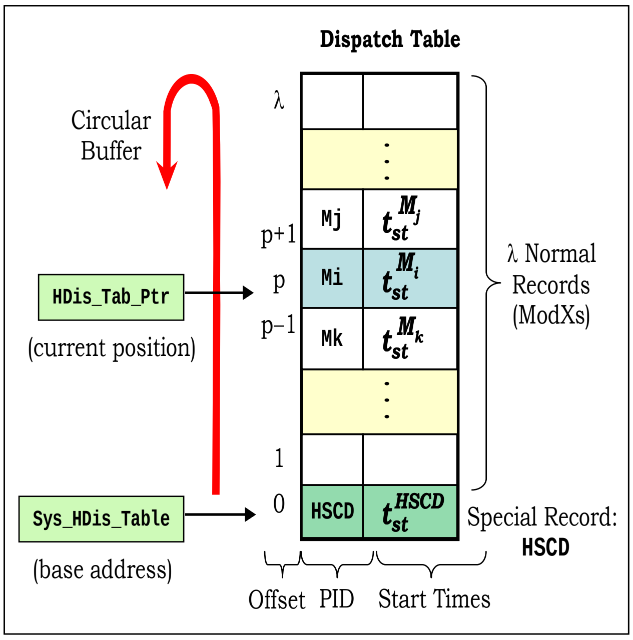

Figure 7. Structure of the Dispatch Table

The Dispatch Table (see Figure 7) is a special circular buffer of length

$$(\lambda + 1) \cdot record\_size \qquad (7)$$

and containing 1 special record (the first one) and λ normal records. The special record identifies the execution of HSCD itself, which will start the next scheduling cycle. The other records describe the schedule of ModXs to be run during the current cycle.

At a given moment, in the Dispatch Table, there can be several records referring to the execution of a particular ModX during the current scheduling cycle, if the ModX period is short enough. Obviously, the start times will be different.

As a result of the non-preemptive approach regarding task scheduling within the HRT context, the start time of a particular ModX in the Dispatch Table complies with the relation:

$$t_{st}^{M_i} \geq t_{st}^{M_k} + t_{ex}^{M_k} = t_{st}^{M_k} + \texttt{Mk.WCET} \qquad (8)$$

where: $t_{st}^{M_i}$ is the start time of the current ModX (M_i), and $t_{st}^{M_k}$ and $t_{ex}^{M_k}$ are the start time and the execution time, respectively, of the ModX previously scheduled in the table (see Figure 7).

All the ModXs referred in the Dispatch Table are in the "*SCD*" state (*Scheduled*).

The HARETICK executive (HDIS) is activated each time an RTC interrupt occurs. HDIS reads the current record in the Dispatch Table (i.e. the ModX scheduled for execution at the current time instance: M_i in Figure 7). HDIS also reads from the table the start time of the next ModX (M_j) and programs the RTC timers to generate an interrupt when that ModX is scheduled (i.e. at $t_{st}^{M_j}$).

HDIS also reads the execution count of the current ModX, and if it has a finite, non-zero value, the ModX will be launched in execution (after decrementing the count). In this case, it enters the "*RUN*" state.

On the other hand, if the execution count is zero, it defines a Ghost ModX, equivalent to the "*GST*" state. Ghost ModXs are not executed by the kernel.

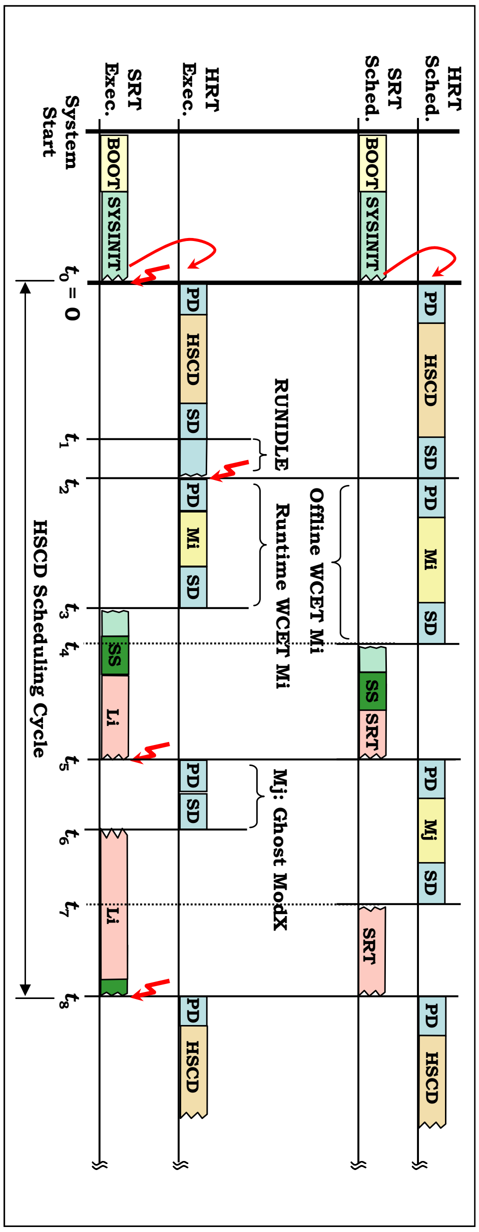

Figure 8. Task scheduling and execution within the HARETICK kernel

Figure 8 depicts an example of application scheduling and execution on the HARETICK kernel, within its two operating contexts: HRT (for ModXs) and SRT.

The HRT context is launched after system startup and initialization (t_0), using the only interrupt allowed, the RTC interrupt. First, the prefix component of the HRT executive (PD) saves the SRT context and prepares the scheduler (HSCD) for execution, which, in turn, creates the list of ModXs (M_i, M_j) to be run during the current scheduling cycle. At termination, every ModX calls the executive suffix (SD), which decides whether to restore the SRT context and hand over the control (t_3), if there is enough time remaining until the next execution of a scheduled ModX. If not, SD waits to be interrupted by the RTC (the RUNIDLE state, started at instance t_1 and interrupted at t_2).

The SRT tasks (L_i, etc) are scheduled and executed in a traditional, time-sharing and priority-based manner, by the SRT scheduler (SS).

Figure 8 also depicts the behavior of the kernel in the case of a Ghost ModX (M_j, scheduled a t_3): PD calls directly the PD component of the HRT executive.

Conclusion

This paper discusses the problem of designing and implementing a real-time compact kernel for embedded and DSP-based platforms, able to provide a fully predictable execution environment for critical applications. Based on a sound and uniform set of models defined for time, signals and tasks, we describe the architecture and operating principles of a particular real-time kernel – "HARETICK", which is designed around the key idea of enabling concurrent execution of HRT and SRT tasks in two separate contexts. The HRT execution context is based on non-preemptive scheduling algorithms and has precedence over the SRT context, which uses traditional, preemptive, priority-based scheduling techniques.

Considering our approach from another perspective, the entire concept, mechanisms and structures used to support the HRT context in the HARETICK kernel, can be used as an extension to classical operating systems, which are based on time-sharing, preemptive task execution (equivalent to the SRT context), thus providing them with the capability of guaranteeing the temporal behavior needed by the HRT tasks of a particular application. The operations required to adapt a traditional system to the HRT context extension include setting the highest priority to the RTC interrupt and blocking all other interrupts during HRT executions.

On the other hand, the SRT context helps also to overcome the system's drastic lack of efficiency when running the HRT context. The scheduling and execution mechanisms for the HRT tasks rely on pessimistic assumptions and evaluations of execution time and signal interaction (periodic polling). In the actual operating conditions of the system, these conditions have a small probability of occurrence, thus leading to many time intervals in which the system is idle. SRT tasks will then be executed.

Currently, the HARETICK kernel is partially developed and tested with good results on a Motorola DSP56307 platform. The HRT executive (HDIS) has been fully implemented and a preliminary version of scheduler (HSCD) has been used to test the system with simple sets of ModXs. All the tests proved a correct behavior of the HRT applications with respect to their temporal specifications.

References

[1] V. Cretu., T. Jurca, M. V. Micea, I. Sora, (2003) *Instrumentation and Measurement in Romania: Technical Developments at 'Politehnica' University of Timisoara*, in IEEE Instrumentation & Measurement Magazine, Vol. 6, No. 3, pp. (41-47), September.

[2] M. V. Micea, M. Stratulat, D. Ardelean, D. Aioanei, (2001) *Implementing Professional Audio Effects with DSPs*, in Transactions on Automatic Control and Computer Science, Vol. 46 (60), Periodica Politehnica, Timisoara, pp. (55-60).

[3] J. A. Stankovic, (1992) *Real-Time Computing,* Invited paper, BYTE, pp. (155-160), August.

[4] J. A. Stankovic, (1992) *Distributed Real-Time Computing: The Next Generation*, Invited keynote paper, Special issue of *Journal of the Society of Instrumentation and Control Engineers of Japan*, Vol. 31, No. 7, pp. (726-736).

[5] R. Chapman, (1994) *Program Timing Analysis*, Technical Report, Dependable Computing Systems Centre, University of York.

[6] K. Ramamritham, J. A. Stankovic, (1994) *Scheduling Algortihms and Operating Systems Support for Real-Time Systems*, in Proceedings of the IEEE, Vol. 82, No. 1, pp. (55-67), January.

[7] D. B. Stewart, (2001) *Twenty-five Most Common Mistakes with Real-time Software Development*, in 2001 Embedded Systems Conference, Class 270, San Francisco, April.

[8] K. Jeffay, D. Stanat, C. Martel, (1991) *On Non-Preemptive Scheduling of Periodic and Sporadic Tasks*, in Proceedings of the 12th IEEE Real-Time Systems Symposium, San Antonio, Texas, IEEE Computer Society Press, pp. (129-139), December.

[9] L. George, N. Rivierre, M. Spuri, (1996) *Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling*, Rapport de recherche, Nr. 2966, Institut National de Recherche en Informatique et en Automatique, INRIA, Rocquencourt, France, September.

[10] D. Chen, A. Mok, S. Baruah, (1998) *On Modeling Real-time Task Systems*, Lecture Notes in Computer Science, No. 1494, Springer-Verlag, pp. (153-169), October.

[11] P. Bellini, R. Mattolini, P. Nesi, (2000) *Temporal Logics for Real-time System Specification*, ACM Computing Surveys, Vol 32, No. 1.

[12] Motorola, Inc., (2000) *DSP56300: 24-Bit Digital Signal Processor: Family Manual*, Rev. 3, DSP56300FM/AD, Semiconductor Products Sector, DSP Division, Austin, USA, November.

[13] Motorola, Inc., (1998) *DSP56307: 24-Bit Digital Signal Processor: User's Manual*, DSP56307UM/D, Revision 0, 08/10/98, SPS, DSP Division, Austin, USA, August.